Records

```
class Person {
    String name;
    int age;
    String city;
    String nationality;
    int ssn; }
```

Operations with records

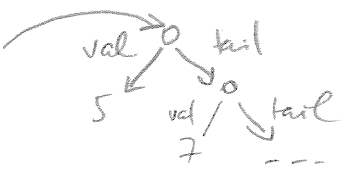
- declaration of record type
- declaration of record vars
  (= pointers to records)
- creation of records
  (= allocation of storage)
- selection of fields

List definition in Java

```
class List {
    int value;
    List tail; }
```

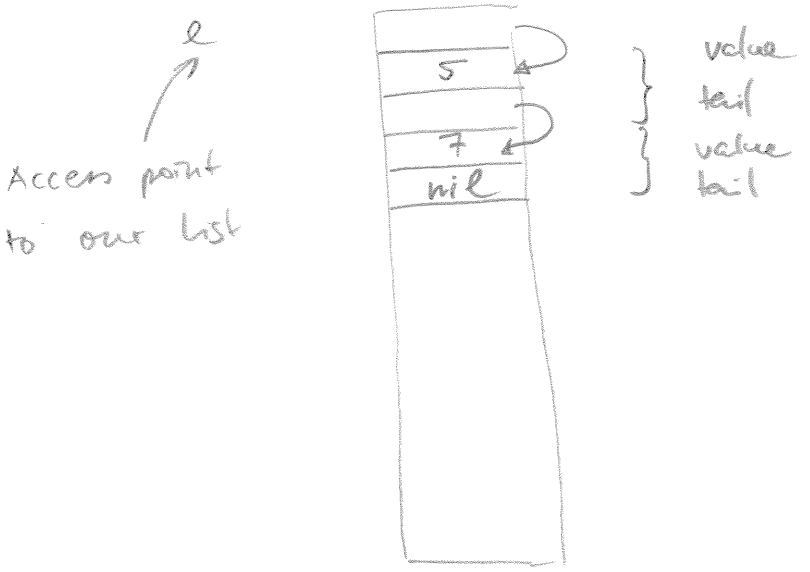
List are a recursive data structure

List l = new List();



How much space?

Pointers



We need type information to interpret the storage content correctly

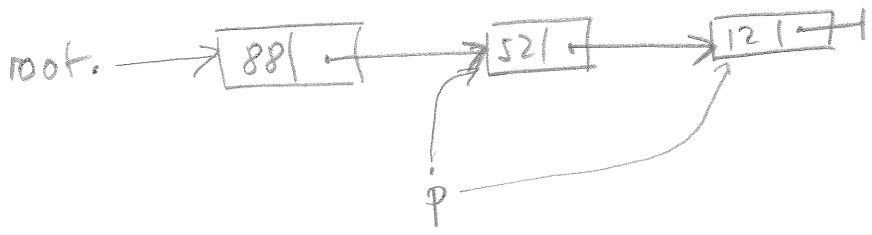
Java

object = pointer + storage space

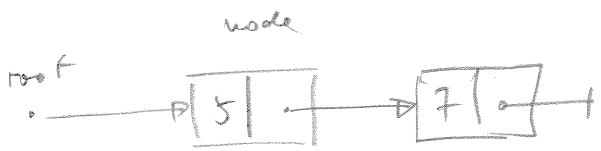
l.value : no explicit dereferencing

garbage collection

Populating a list w/ integers

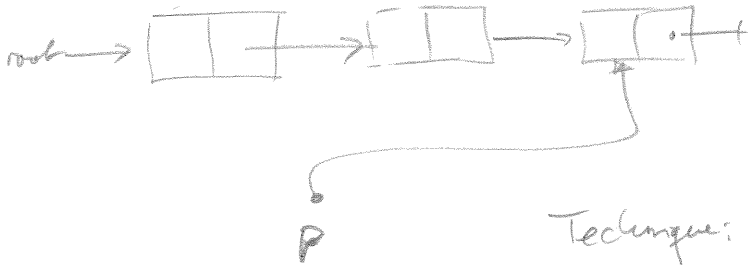


Dynamic data structures: List



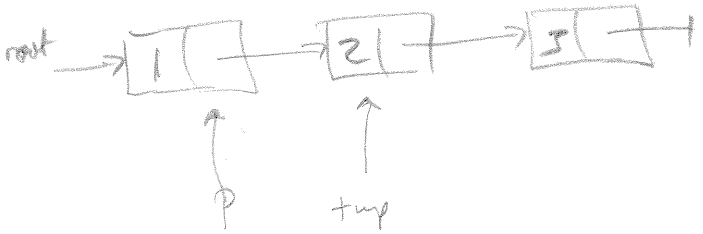
Operations on lists

- Creating a list with values
- Traversal
- Insertion
  - at the beginning
  - at the end



Technique: trailing pointer

- Deletion of an element (2)

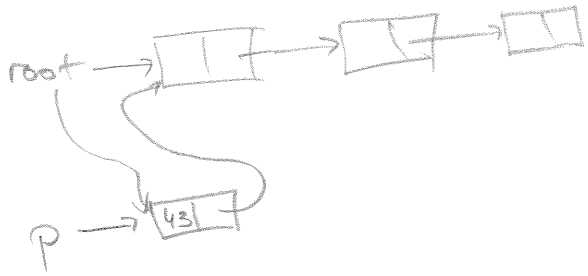


p.next = tmp.next

List traversal

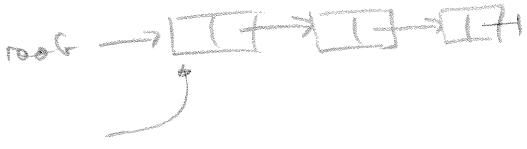
```
node p;
p = root;
while (p != null) {
        " print p.val";
        p = p.next; }
```

List insertion




```
node p = new node();
p.val = 43;
p.next = root;
root = p;
```

At the end



Try:

```
node q = root;
while (q != null) {
    q = q.next; };
```

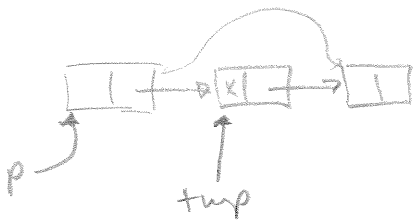
```
if (root == null){
        root = new node();
        root.val = 43;
        root.next = null;
} else{
    q = root;
    while (q.next != null) { q = q.next;};
    q = new node();
    q.val = 43;
    q.next = null;
}
```

new node(43, null);

new node(43, null);

List deletion : delete x

```
p = root;
if (p.val==x){
        root = p.next;
} else{
    while (p.next != null && p.next.val != x) {
            p = p.next;
    }
    tmp = p.next;
    p.next = tmp.next;
}
```

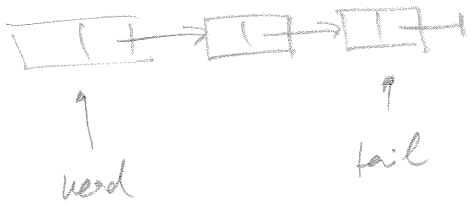


Lists : Cost of operations

- insertion
  - at beginning $\Theta(1)$
  - at end $\Theta(n)$
- deletion
  - at beginning $\Theta(1)$
  - at end $\Theta(n)$
- traversal $\Theta(n)$
- search $\Theta(n)$

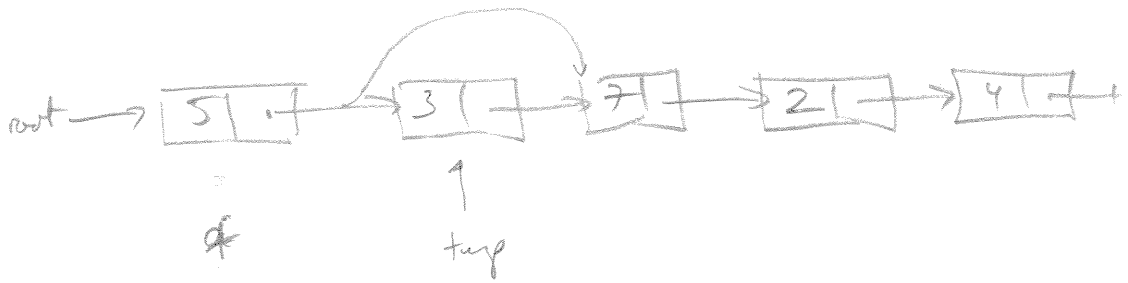
Lists : Variants

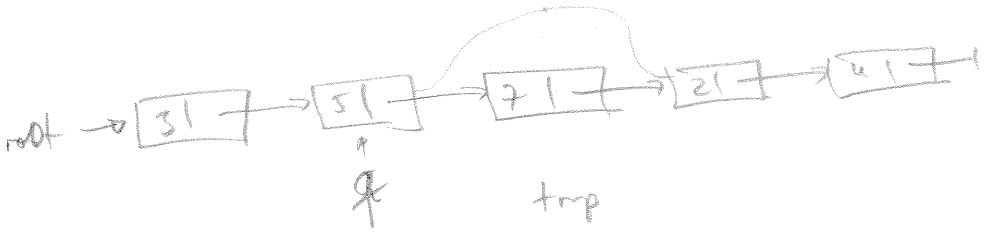
head, tail pointers

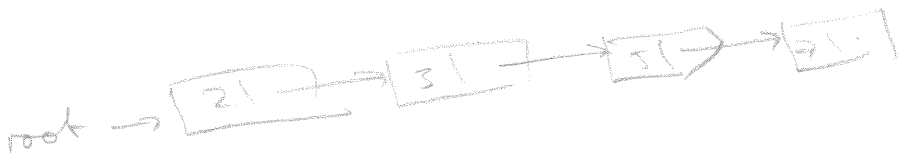


Exercise: Implement insertion sort using lists?

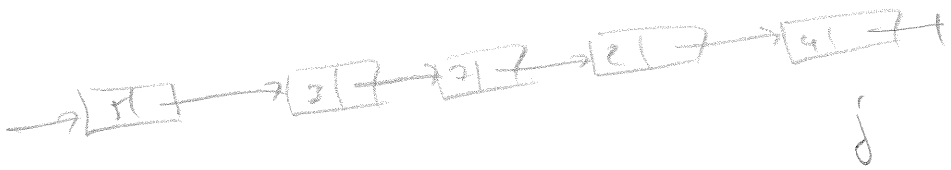
— Idea :







Exercise: Can we implement Quicksort?



x := 4

We can move from left to right,
but not from right to left!

Doubly Linked Lists

```
class node {
    int val;
    node next, prev;
}
```

Algorithms to implement Queue

Abstract Data Types

- operations
- axioms, laws

Example: Stack (of int): algebraic specification style

init : → Stack
push : Stack × int → Stack
isEmpty : Stack → boolean
top : Stack → int
pop : Stack → Stack

isEmpty (init, ()) = true
isEmpty (push (s, u)) = false (f.a. s: Stack, u: int)

top (push (s, u)) = u

pop (push (s, u)) = s

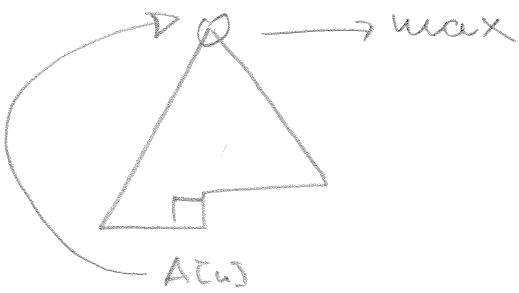
pop (s)
- precondition: isEmpty (s)
- postcondition: top(s) is removed from s

size: Stack → ℕ
size (init, ()) = 0
size (push (s, u)) = size (s) + 1

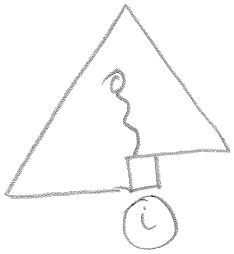
Priority Queue

Operation	Sorted List	Heap
Insert (S, x)	$\Theta(n)$	$\Theta(\log n)$
Extract max	$\Theta(1)$	~~$\Theta(n)$~~ $\Theta(\log n)$

ExtractMax(A)



Insert (A, k)



Insert(A, 15)